AGiDS: A Grid-based Strategy for Distributed Skyline Query Processing

João B. Rocha-Junior*, Akrivi Vlachou**, Christos Doulkeridis**, and Kjetil Nørvåg

Norwegian University of Science and Technology
{joao,vlachou,cdoulk,noervaag}@idi.ntnu.no

Abstract. Skyline queries help users make intelligent decisions over complex data, where different and often conflicting criteria are considered. A challenging problem is to support skyline queries in distributed environments, where data is scattered over independent sources. The query response time of skyline processing over distributed data depends on the amount of transferred data and the query processing cost at each server. In this paper, we propose AGiDS, a framework for efficient skyline processing over distributed data. Our approach reduces significantly the amount of transferred data, by using a grid-based data summary that captures the data distribution on each server. AGiDS consists of two phases to compute the result: in the first phase the querying server gathers the grid-based summary, whereas in the second phase a skyline request is sent only to the servers that may contribute to the skyline result set asking only for the points of non-dominated regions. We provide an experimental evaluation showing that our approach performs efficiently and outperforms existing techniques.

1 Introduction

Skyline queries [1] have attracted much attention recently, mainly because they help users to make intelligent decisions over data that represent many conflicting criteria. The skyline of a given set of d-dimensional points S is the set of points which are not dominated by any other point of S. A point p is dominated by another point q if and only if q is not worse than p in any dimension and q is better than p in at least one dimension.

During the last decades, the vast number of independent data sources and the high rate of data generation make central assembly of data at one location infeasible. As a consequence, data management and storage become increasingly distributed. Skyline query processing in a distributed and decentralized environment has received considerable attention recently [2–12]. Distributed information systems are applications that can benefit from this query type. Consider a hotel

* Ph.D. student at NTNU, on leave from Universidade Estadual de Feira de Santana.
** This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme.

reservation system, consisting of a large set of independent servers geographically dispersed around the world, each of them storing its own data about hotels. Such a system could potentially provide booking services over the universal hotel database, by allowing users to find suitable hotels based on skyline queries.

In this paper, we propose a novel way to process skyline queries efficiently in a distributed environment. We make no assumption on the existence of an overlay network that connects servers in an intentional manner, thus a querying server directly communicates with other servers. We assume horizontal partitioning of data to servers. Furthermore, each server stores a lightweight data structure, which enables obtaining summary information about the data stored at each server. Each participant is responsible to maintain its own data structure with information about its local data. Based on this information, we are able to process skyline queries only at the servers and regions in the d-dimensional data space, which have data points belonging to the skyline result set. By means of an experimental evaluation, we show that our solution reduces both response time through higher parallelism and the amount of data transferred.

This paper makes the following contributions: First, we present an overview of the current research on distributed skyline query processing. Then, we propose a new strategy to compute skyline queries efficiently in a distributed environment, where data is horizontally distributed to servers and no overlay network exists. Further, we propose the use of a data structure to maintain summary information about the data stored at each server. Finally, we perform an experimental evaluation in order to demonstrate the effectiveness of our approach.

This paper is organized as follows: Section 2 presents an overview of the related work. In Section 3, we provide the necessary preliminaries and definitions. In Section 4, we describe our approach for distributed skyline computation. The experimental evaluation is presented in Section 5 and we conclude in Section 6.

2 Related Work

Skyline computation has recently attracted considerable attention both in centralized [1] and distributed domains [2–12]. One of the first algorithms in the distributed domain, by Balke *et al.* [2], focuses on skyline query processing over multiple distributed sources, with each source storing only a subset of attributes (vertical data distribution). Later, most of the related work has focused on highly distributed and P2P environments, assuming all sources store common attributes (horizontal data distribution). In the following, we provide a brief survey of existing P2P skyline processing algorithms. A comparative overview of distributed skyline literature is presented in Table 1.

Approaches that assume horizontal data distribution can be classified in two main categories. In the first category, the proposed methods assume space partitioning among peers, thus each peer is responsible for a disjoint partition of the data space. Towards this goal, a structured P2P or tree-based network overlay is employed. The system controls the location of each data point and splits the data in a way that the system can visit first the peers with higher probability of

Papers	P2P overlay	Skyline query type	Partitioning
MANETs [4]	no overlay	global	data
QTree [7]	unstructured	approximate	data
DSL [5]	DHT (CAN)	constrained	space
SKYPEER [3]	super-peer	subspace	data
BITPEER [8]	super-peer	subspace	data
PaDSkyline [6]	no overlay	constrained	data
iSky [9]	BATON	global	space
SkyFrame [10, 11]	CAN and BATON	global, approximate	space
FDS [12]	no overlay	global	data

Table 1. Summary of approaches for P2P skyline processing.

having skyline points. DSL was proposed by Wu *et al.* [5] and it is the first paper that addresses constrained skyline query processing over disjoint data partitions, which are assigned to peers using CAN. Wang *et al.*[10] propose the SSP algorithm based on the use of a tree-based overlay (BATON) for assigning data to peers. They use a one-dimensional mapping of data based on z-order and then data is assigned to peers. Later, the authors present SkyFrame [11] as an extension of their work. SkyFrame is a framework that comprises two methods: GSS (Greedy Skyline Search) and RSS (Relaxed Skyline Search). GSS and RSS can run on top of either CAN or BATON. GSS achieves low bandwidth consumption, whereas RSS reduces the overall response time. Chen *et al.* [9] propose the iSky algorithm, which employs another transformation, namely iMinMax, in order to assign data to BATON peers.

In the second category, data partitioning is assumed and each peer autonomously stores its own data. Huang *et al.* [4] study skyline query processing over mobile ad-hoc networks. The aim is to reduce communication costs as well as processing costs on the individual device. In SKYPEER [3], each super-peer computes and stores the extended skyline of its associated peers. Then, when a super-peer processes a skyline query, the query is forwarded to neighboring super-peers, processed locally using the extended skyline set, followed by in-network merging of results. Fotiadou *et al.* [8] propose BITPEER that uses a bitmap representation, in order to improve the performance of query processing. Hose *et al.* [7] use distributed data summaries (QTree) about the data stored by the peers. During skyline query processing, the QTree is used as routing mechanism to determine the peers, which need to be processed, in order find the skyline points. The approach supports approximate result sets for reducing the processing cost and provides guarantees for the completeness of the result.

Cui *et al.* [6] study skyline query processing in a distributed environment, without the assumption of an existing overlay network, where a coordinator can directly communicate with all peers (servers). They propose the use of MBRs (Minimum Bounding Regions) to summarize the data stored at each server. The proposed PaDSkyline algorithm works in two steps. In the first step, the MBRs

of all servers are collected and assigned to incomparable groups, which can be queried in parallel, while specific plans are used within each group. Subsequently, servers are queried and the results are returned back to the coordinator. Recently, in [12], a feedback-based distributed skyline (FDS) algorithm is proposed, which also assumes no particular overlay network. The algorithm is efficient in terms of bandwidth consumption, however it requires several round-trips to compute the skyline, thus it may incur high response time.

Our framework, similar to the ones presented in [6, 12], assumes no particular network topology. In contrast to [12], our approach requires only two round trips, thus avoiding high latency caused by multiple round trips. Differently than [6], we use grid-based summary information instead of MBRs. Therefore, the data space is split into non-overlapping regions, which makes easier to identify and discard dominated regions across different servers, whereas MBRs may have high overlap and cannot be discarded.

3 Preliminaries

Given a data space D defined by a set of d dimensions $\{d_1, ..., d_d\}$ and a dataset P on D with cardinality $|P|$, a point $p \in P$ can be represented as $p = \{p_1, ..., p_d\}$ where p_i is a value on dimension d_i. Without loss of generality, let us assume that the value p_i in any dimension d_i is greater or equal to zero ($p_i \geq 0$) and that for all dimensions the minimum values are more preferable. Figure 1 offers an overview of the symbols that are used throughout this paper.

Definition: *Skyline Query (SQ).* A point $p \in P$ is said to *dominate* another point $q \in P$, denoted as $p \prec q$, if (1) on every dimension $d_i \in D$, $p_i \leq q_i$; and (2) on at least one dimension $d_j \in D$, $p_j < q_j$. The *skyline* is a set of points $SKY_P \subseteq P$ which are not dominated by any other point in P. The points in SKY_P are called skyline points.

Consider a database containing information about hotels where each tuple of the database is represented as a point in a data space consisting of different characteristics of the hotel. In our example, the y-dimension represents the price of a room, whereas the x-dimension captures the distance of the hotel to a point of interest such as the beach (Figure 2). According to the dominance definition, a hotel dominates another hotel because it is cheaper and closer to the beach. Thus, the skyline points are the best possible tradeoffs between price and distance. In our example, the hotels that belong to the skyline are a, i, m and k.

In this work, we assume a set of N servers S_i participating in the distributed skyline computation. Each server S_i stores a set of points P_i that is a fraction of the dataset P such that $P_i \subseteq P$ and $\bigcup_{1 \leq i \leq N} P_i = P$. The data partitions are not necessarily disjoint and they may overlap. We also assume that each server S_i can directly connect to any other server S_j. Thus, we do not make an explicit assumption about the availability of an overlay network that assigns regions of the data space to specific peers intentionally. Moreover, each server S_i is able

Symbol	Description
d	Dimensions
P	Dataset
$\|P\|$	Cardinality of the dataset
p, q	Data points
S_i	Server i
N	Number of servers
P_i	Dataset of server S_i
SKY_P	Skyline points of dataset P

Fig. 1. Overview of symbols.

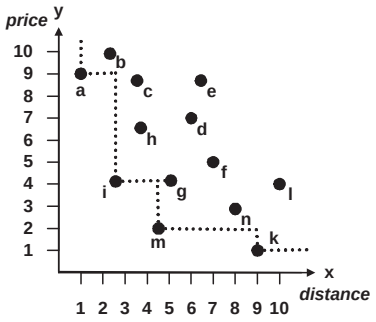


Fig. 2. Skyline example.

to compute the local skyline set SKY_{P_i} (mentioned also as local skyline points) based on the locally stored points P_i. Similarly, we refer to the skyline set SKY_P of the dataset P as global skyline set.

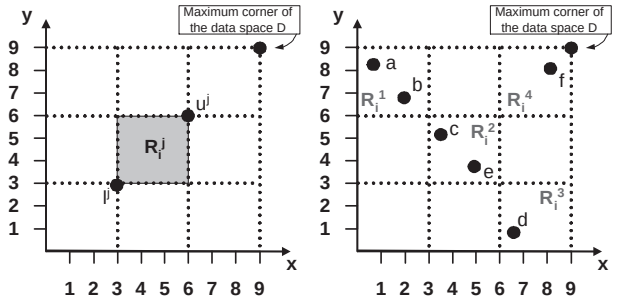
Observation: A point $p \in P$ is a skyline point $p \in SKY_P$ if and only if $p \in SKY_{P_i} \subseteq P$ and p is not dominated by any other point $q \in SKY_{P_j}, j \neq i$.

In other words, the skyline points over a horizontally partitioned dataset are a subset of the union of the skyline points of all partitions. Therefore, aggregating and merging the local skyline points produces the skyline result set SKY_P. In the following, we propose an algorithm that computes the exact skyline result set SKY_P by transferring only a subset of the local skyline points.

4 AGiDS Algorithm

In a distributed environment, a skyline query can be initiated by any server, henceforth also called query originator (S_{org}). The naive approach to process a skyline query is to send the query to all the servers S_i, which in turn process the skyline locally, and report the local skyline result to S_{org}. Then, S_{org} merges all received results to obtain the global skyline set. This approach requires transferring an excessive amount of data and processing the complete skyline query at each server. Instead, we present an algorithm (AGiDS) that improves the overall performance, aiming at reducing both processing and communication cost.

AGiDS is divided in two phases: planning and execution. To this end, we propose the usage of a grid-based data structure that captures the data distribution of each server (Section 4.1). During the planning phase (Section 4.2), S_{org} contacts all servers S_i and obtains information about which regions contain data that belong to the local skyline result set. The assembled regions enable two improvements during the subsequent execution phase (Section 4.3): S_{org} queries only the servers that contain at least one non-dominated region, and more importantly S_{org} requests only a subset of local skyline points, namely



(a) Grid structure at S_i. (b) Dominations of regions.

Fig. 3. Information at P_{org} after receiving the important regions from all the servers.

the local skyline points that belong to non-dominated regions. After gathering the relevant points, S_{org} computes the skyline result set by merging only the necessary regions.

4.1 Grid-based Data Summary

AGiDS employs a grid-based data summary for describing the data available at each server. We assume that all servers share a common grid and the partitions cover the entire universe D. Optimizing the grid is out of the scope of this paper and in the following we assume that the partitions are created by splitting each dimension d_i in a predefined number of slices. Therefore, each partition has the form of a d-dimensional hypercube. In the following, we refer to a partition of the server S_i as the j-th data region R_i^j of server S_i. Obviously, the regions of a server S_i are non-overlapping d-dimensional data regions R_i^j. A region R_i^j is defined by two points l^j and u^j indicating the lower left and upper right corner of the region respectively.

Consider for example Figure 3(a), that depicts the data structure maintained by each server S_i and a data region R_i^j. We define a region of R_i^j of server S_i as *populated*, if there exists at least one point $p \in P_i$ enclosed in region R_i^j, i.e. for each dimension $d_i \in D$, $l_i^j \leq p_i < u_i^j$. For instance, in Figure 3(b), the populated regions are R_i^1, R_i^2, R_i^3 and R_i^4. Furthermore, given two populated regions of R_i^j and R_i^k, we define the following dominance relationships. Notice that all the examples used in the definitions below can be found at Figure 3(b).

Definition: R_i^j *dominates* R_i^k, if the right upper corner u^j of R_i^j dominates the left lower l^k corner of R_i^k. For example, R_i^2 dominates R_i^4, which means that any point of R_i^2 dominates all the points of R_i^4. Notice that point f is dominated by both points c and e of region R_i^2.

Definition: R_i^j *partially dominates* R_i^k, if R_i^j does not dominate R_i^k, but the left lower corner l^j of R_i^j dominates the right upper corner u^k of R_i^k. For example,

R_i^1 partially dominates R_i^4. Therefore, some points of R_i^1 may dominate some or all points of R_i^4. In our example point b dominates point f, but a does not dominate f.

Definition: R_i^j and R_i^k are *incomparable*, if there is no point in R_i^j able to dominate any point in R_i^k and vice versa, e.g. R_i^1 and R_i^2. This means that R_i^j does not dominate nor partially dominate R_i^k, and also R_i^k does not dominate nor partially dominate R_i^j.

Finally, we define as *region-skyline* of a server S_i the skyline of the populated regions of S_i, based on the aforementioned definition of region dominance. Thus, the region-skyline contains all regions that are not dominated by another region. Moreover, it can be easily computed in a way similar to the traditional skyline query on the data points. In Figure 3(b), for example, the regions R_i^1, R_i^2 and R_i^3 define the region-skyline of peer S_i in the data space D.

Using the grid-based data summary common for all servers enables determining regions that are dominated by other servers in an efficient way. Also, the grid regions are easily and uniquely identified with a small cost. Furthermore, local skyline points can be easily merged based on the region that they belong. Having a common grid makes the regions directly comparable, which means that each server can uniquely identify a region in the entire universe D. Moreover, a compact representation of regions is feasible, usually just a small number of bits, which avoids the use of two d-dimensional points. Thus, each server S_i just needs to send an identifier of the populated regions R_i^j.

4.2 Phase 1: Planning

The planning phase starts with S_{org} requesting the region-skyline of each server S_i in parallel. We assume that S_i has a grid-based data structure, which is used to answer queries of S_{org}. The message sent by S_i back to S_{org} returns the region-skyline of S_i. In Figure 4, for example, B is the single region sent to S_{org} by S_2, because all other populated regions in S_2 are dominated by region B and are not part of the region-skyline of S_2.

After receiving the region-skyline from all the servers, S_{org} computes the global region-skyline. The global region-skyline is the skyline of the populated regions of all servers contacted by S_{org}. The objective of computing the global region-skyline is eliminating all the regions of a server, which are dominated by other regions of another server. For example, in Figure 4, region H is dominated by region A and does not belong to the global region-skyline. Computing the global region-skyline is much cheaper than computing the entire skyline over data points, because the number of populated regions is much smaller than the number of points. After computing the global region-skyline, S_{org} has information about all the regions in the entire data space D, which have points that may be part of the global skyline. In the current example, S_{org} computes as global region-skyline the regions A, B and F.

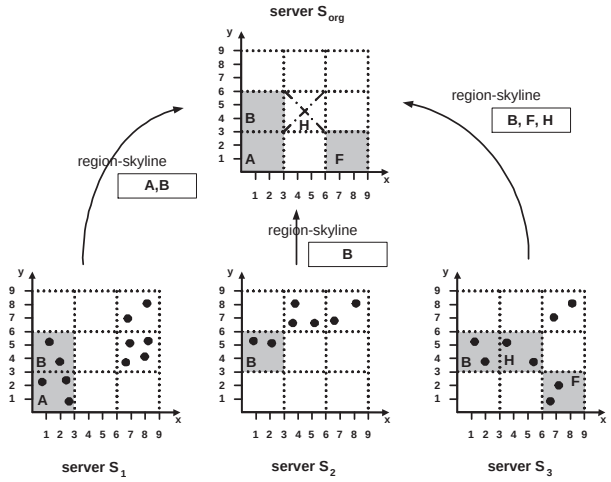


Fig. 4. Planning phase of AGiDS.

4.3 Phase 2: Execution

The execution phase starts after S_{org} has computed the global region-skyline. Then, S_{org} can determine the servers that may contribute to the global skyline result set. Furthermore, S_{org} knows which regions of each server S_i are necessary for the skyline computation. Therefore, before sending a skyline query to a server S_i, S_{org} attaches the identifiers of the regions that need to be examined for the query. For example, in order to process the skyline query at the server S_3, only the skyline points at the regions B and F are requested, since region H was eliminated in the previous phase.

Thus, a server S_i receives a message requesting the skyline of the regions that are relevant for the global skyline. S_i processes the skyline of the regions requested, discarding points dominated by points of other requested regions. Then, S_i reports its result to S_{org}, which comprises the local skyline points of the requested regions, after discarding the dominated points.

The local skyline points returned to the S_{org} are grouped based on the region they are enclosed in. S_{org} receives the points of each server S_i and merges them into the global skyline. Merging of the local skyline points is performed efficiently based on the information of the regions. Only the points enclosed into regions which are partially dominated have to be merged. Therefore, S_{org} is able to return immediately to the user the points enclosed in a region R_i, if there is no other server which has a region R_j that partially dominates R_i. Thus, the skyline points are reported progressively by S_{org} to the user.

Parameter	**Values**
Dimensions	2, **3**, 4, 5
Number of servers	**50**, 100, 150, 200, 250
Cardinality of each server	10K, **25K**, 50K, 75K, 100K
Network speed	0.2 Mbit/s
Filter points percentage	10%
Maximum number of regions	1024

Table 2. Settings used in the experiments.

5 Experimental Evaluation

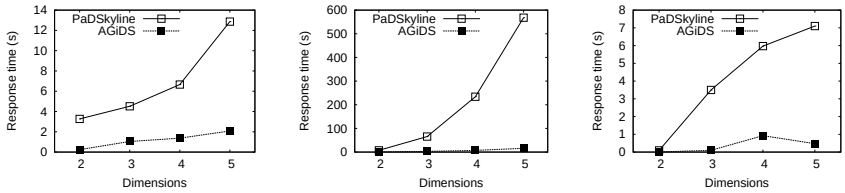
In this section, we study the performance of our distributed processing strategy in a simulated environment. For this purpose, we compare our approach against the PaDSkyline algorithm proposed by Cui *et al.* [6]. Both approaches were implemented in Java, while the network aspects were simulated using DesmoJ[1], an event-based simulator framework. We consider two main performance aspects: (1) the response time and (2) the total amount of data transferred. The experiments were conducted in a 3GHz Dual Core AMD processor equipped with 2GB RAM.

We compare our approach against PaDSkyline, because both approaches assume no overlay network and S_{org} can directly communicate with all servers. Furthermore, both approaches have a first phase in which some summary information about all the servers is collected. This information is used to define the strategy adopted on the second phase, where the skyline points are collected. In the following, we explain the implementation of PadSkyline in more detail.

After collecting the MBRs of all the servers, PaDSkyline identifies incomparable groups of MBRs (and servers) and processes each group of servers in parallel. Within each group, servers are organized in a sequence for processing the query, thus a way to order the servers is necessary. The servers with MBRs closer to the minimum corner of the universe are accessed first. At each server, the local skyline is computed, using filter points received from the previous server. The filter points are selected based on maximizing the dominated volume (maxSum heuristic in [6]) and they are used to reduce the amount of data transferred.

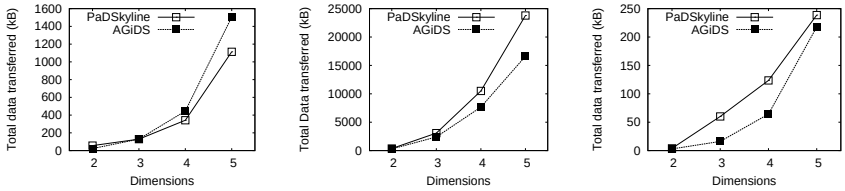
Then, the local sever computes the new filter points, which are a percentage of the points of the local skyline. We use the default value of 10%, as in [6]. The servers in the group that are dominated by the new filter points are discarded. After that, the query is forwarded to the next server with the new filter points attached. The process within each group finishes when the group becomes empty. This is the *linear approach* for intra-group processing, as described in [6].

[1] http://desmoj.sourceforge.net/home.html



(a) Uniform data. (b) Anti-correlated data. (c) Correlated data.

Fig. 5. Evaluation of response time for various data distributions.



(a) Uniform data. (b) Anti-correlated data. (c) Correlated data.

Fig. 6. Evaluation of the total data transferred for various data distributions.

5.1 Experimental Settings

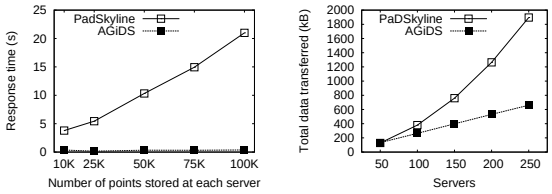
In the experiments, we have used three types of synthetic datasets: uniform, correlated, and anti-correlated. The settings used in the simulator are listed in the Table 2. Unless mentioned explicitly, we have used as default setup a 3-d dataset, distributed to 50 servers with each server storing 25K points.

The network communication was simulated based on events; therefore, the time required to transfer data is computed by dividing the size of the data transferred with the network speed. Filter points are used only in the simulation of PaDSkyline, thus the percentage of filter points applies only for PaDSkyline. Similarly, the maximum number of regions R_{max} is used only in the simulation of AGiDS. This number is used to compute the number of slices s of each dimension of the grid, as: $s{=}R_{max}^{1/d}$, where d is number of dimensions.

5.2 Experimental Results

The response time achieved with AGiDS is better than the response time obtained with PaDSkyline in all evaluated setups, as shown in Figure 5. Figures 5(a), 5(b), and 5(c) provide comparative results for uniform, anti-correlated and correlated datasets respectively. We study the scalability of the algorithms with increasing dimensionality d. In all cases, the response time increases more rapidly in PaDSkyline than in AGiDS. In particular, notice the excessive response time required by PaDSkyline for the anti-correlated dataset.

The main reason for the better response time achieved by AGiDS is the higher parallelism during the second phase. After collecting the region-skyline,



(a) Cardinality of P_i. (b) Number of servers.

Fig. 7. Scalability study of AGiDS.

AGiDS processes in parallel the skyline points at regions of each server which can contribute to the global skyline. In contrast, the linear approach of PaDSkyline achieves parallelism in the second phase only between incomparable groups. During the intra-group processing, PaDSkyline uses sequential execution on servers, thereby spending much time to produce the global skyline result.

Another reason for achieving better response time with AGiDS is the reduced processing time at each server. While PaDSkyline computes the entire skyline, AGiDS computes only the skyline at the regions requested by S_{org}, which are the only regions that can contribute to the global skyline.

Figure 6 depicts the total amount of transferred data for the two algorithms, for different data distributions. This amount corresponds to the number of bytes transferred during query processing for all required communications. Only in the case of uniform data (Figure 6(a)) PaDSkyline transfers less data than AGiDS. Both in the anti-correlated (Figure 6(b)) and correlated (Figure 6(c)) data distributions, AGiDS performs better. PaDSkyline transfers less data in the case of uniform data distribution, because the MBRs of the servers are completely overlapping, and one filter point located near the minimum corner can prune several servers within a group. However, notice that despite the higher amount of transferred data in the case of the uniform dataset, AGiDS achieves much better response time. In any case, for anti-correlated and correlated datasets, AGiDS is better in both measures.

Then, in Figure 7(a), we test the effect of increasing the cardinality of the data stored at each server from 10K to 100K points, following a uniform data distribution. We observe that the response time of AGiDS is not affected by the increasing number of points stored at each server. The main reason is that the number of regions in the region-skyline is the same, irrespective of the number of points stored. Therefore, the performance of the planning phase is not affected. Only the number of local skyline points transferred at the second phase changes slightly, but this does not have a significant impact on the response time. However, notice that the response time of PaDSkyline increases significantly with the number of points stored at each server.

Finally, in Figure 7(b), we conduct a scalability study with the number of participating servers. In this setup, AGiDS benefits from the fact that it requires only two steps to finalize the query processing, while PadSkyline may have to access many servers within a group sequentially. Therefore, the amount of data transferred by PaDSkyline grows much faster than with AGiDS.

6 Conclusions

Distributed skyline query processing poses inherent challenges, due to the distribution of content and the lack of global knowledge. In this paper, we presented AGiDS, a skyline query processing algorithm for distributed environments. AGiDS processes skyline queries efficiently by applying a two phase approach. In the first phase, it uses a grid-based data summary on each server, in order to discard regions that cannot contribute to the skyline result. In the second phase, local skyline points that belong to non-dominated regions are selectively collected, in order to produce the global skyline. By means of an experimental evaluation, we have demonstrated the superiority of AGiDS compared to an existing approach that is appropriate for our context.

References

1. Börzsönyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Int. Conf. on Data Engineering (ICDE). (2001) 421–430
2. Balke, W.T., Güntzer, U., Zheng, J.X.: Efficient distributed skylining for web information systems. In: Int. Conf. on Extending Database Technology (EDBT). (2004) 256–273
3. Vlachou, A., Doulkeridis, C., Kotidis, Y., Vazirgiannis, M.: SKYPEER: Efficient subspace skyline computation over distributed data. In: Int. Conf. on Data Engineering (ICDE). (2007) 416–425
4. Huang, Z., Lu, C.S.J.H., Ooi, B.C.: Skyline queries against mobile lightweight devices in MANETs. In: Int. Conf. on Data Engineering (ICDE). (2006) 66
5. Wu, P., Zhang, C., Feng, Y., Zhao, B.Y., Agrawal, D., Abbadi, A.E.: Parallelizing skyline queries for scalable distribution. In: Int. Conf. on Extending Database Technology (EDBT). (2006) 112–130
6. Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y.: Parallel distributed processing of constrained skyline queries by filtering. In: Int. Conf. on Data Engineering (ICDE). (2008) 546–555
7. Hose, K., Lemke, C., Sattler, K.U.: Processing relaxed skylines in PDMS using distributed data summaries. In: Int. Conf. on Information and Knowledge Management (CIKM). (2006) 425–434
8. Fotiadou, K., Pitoura, E.: BITPEER: Continuous subspace skyline computation with distributed bitmap indexes. In: Int. Workshop on Data Management in Peer-to-Peer Systems (DAMAP). (2008) 35–42
9. Chen, L., Cui, B., Lu, H., Xu, L., Xu, Q.: iSky: Efficient and progressive skyline computing in a structured P2P network. In: Int. Conf. on Distributed Computing Systems. (2008) 160–167
10. Wang, S., Ooi, B.C., Tung, A.K.H., Xu, L.: Efficient skyline query processing on peer-to-peer networks. In: Int. Conf. on Data Engineering (ICDE). (2007) 1126–1135
11. Wang, S., Vu, Q.H., Ooi, B.C., Tung, A.K.H., Xu, L.: Skyframe: A framework for skyline query processing in peer-to-peer systems. VLDB Journal **18**(1) (2009) 345–362
12. Zhu, L., Tao, Y., Zhou, S.: Distributed skyline retrieval with low bandwidth consumption. Transactions on Knowledge and Data Engineering (TKDE), to appear (2009)